Alunos: Cecilia, Ivan e Paulo

Precisamos dividir a lista em outras sub-listas guardando as posições de início meio e fim como parâmetros de ordenação:

```
def mergesort(lista,inicio=0,fim=None)
        if fim is None:
            fim = len(lista)
```

None é um valor que você pode atribuir a uma variável que significa vazio, diferente do **pass** que nao faz nada, ele consegue armazenar algo.

Então criamos as variáveis:

- **lista**, é onde são armazenados
- **início** indica a posição a estar na esquerda
- **meio** servindo para alocar a divisão
- **fim**

fim - inicio > 1
Vai dividir recursivamente o array até se tornar um subarray que possua somente 1 elemento.

Qual a função de um algoritmo recursivo?
A ideia básica consiste em diminuir sucessivamente o problema em um problema menor, até o tamanho que permita resolvê-lo de forma direta, sem recorrer a si mesmo.

Fazendo o processo para estar ordenado de menor para maior com apenas 1 array e combinar 2 subarrays e sucessivamente para depois ter 1 array.

```
        if fim - inicio > 1:
            meio = (fim + inicio) // 2
            Classe.mergesort(lista, inicio, meio)
            Classe.mergesort(lista, meio, fim)
            Classe.merge(lista, inicio, meio, fim)
```

Classe.mergesort (e suas variáveis) realiza a recursividade e divide os elementos da lista.

def merge vai realizar a junção
left = lista[inicio:meio], indica seguir do início até o fim da lista e o **right = lista[meio:fim]** do meio até o fim da lista

```
def merge(lista, inicio, meio, fim):
        left = lista[inicio:meio]
        right = lista[meio:fim]
```

for k in range (inicio,fim)
Vai verificar se quem está no topo da lista da esquerda é menor de quem está no topo da lista da direita, então coloca o valor no lugar indicado do lado esquerdo, se não, adiciona na posição ao lado direito.

Código limpo:

```
for k in range(inicio, fim):
            if i >= len(left):
                lista[k] = right[j]
                j += 1
            elif j >= len(right):
                lista[k] = left[i]
                i += 1
            elif left[i] < right[j]:
                lista[k] = left[i]
                i += 1
            else:
                lista[k] = right[j]
                j += 1
```

Código com comentários:

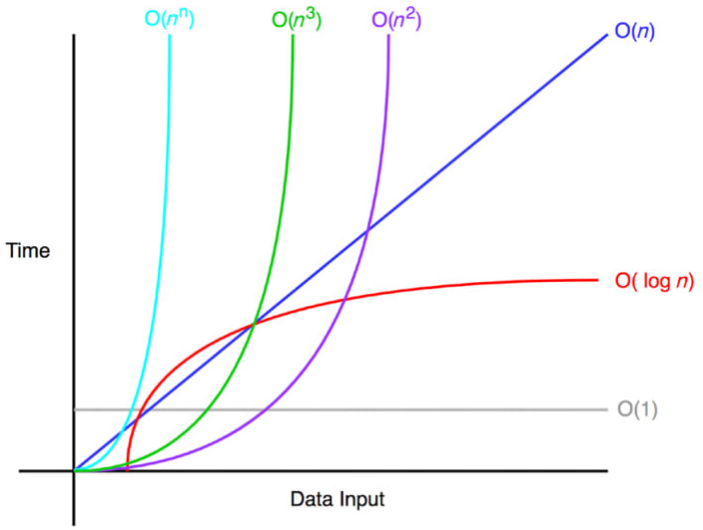
```
for k in range(inicio, fim): # k é que verifica o topo da lista
            if i >= len(left): # se o topo da lista da esquerda for maior do que o topo da lista da direita
                lista[k] = right[j] # adiciona na lista direita na posição j(direita) da lista
                j += 1 # avança a posição do topo da lista da direita
            elif j >= len(right):
                lista[k] = left[i]
                i += 1
            elif left[i] < right[j]:# se o número na posição i (esquerdo) for menor que da posição j (direito)
                lista[k] = left[i] # coloca o número na posição i (esquerdo)
                i += 1
            else:
                lista[k] = right[j] # adiciona o topo da lista na direita da posição a direita
                j += 1
```

Código Completo:

```
def mergesort(lista, inicio=0, fim=None):
        if fim is None:
            fim = len(lista)

        if fim - inicio > 1:
            meio = (fim + inicio) // 2
            Classe.mergesort(lista, inicio, meio)
            Classe.mergesort(lista, meio, fim)
            Classe.merge(lista, inicio, meio, fim)

    def merge(lista, inicio, meio, fim):
        left = lista[inicio:meio]
        right = lista[meio:fim]

        i, j = 0, 0
        for k in range(inicio, fim):
            if i >= len(left):
                lista[k] = right[j]
                j += 1
            elif j >= len(right):
                lista[k] = left[i]
                i += 1
            elif left[i] < right[j]:
                lista[k] = left[i]
                i += 1
            else:
                lista[k] = right[j]
                j += 1

        return lista
```

```
Shell Sort
Tempo para ordenar vetor DESordenado (10k): 0.01296806335449218
Tempo para ordenar vetor DESordenado (25k): 0.04305052757263183
Tempo para ordenar vetor DESordenado (50k): 0.09721064567565918
Tempo para ordenar vetor ORdenado    (10k): 0.01604938507080078
Tempo para ordenar vetor ORdenado    (25k): 0.04101276397705078
Tempo para ordenar vetor ORdenado    (50k): 0.08437180519104004
###############################################
Select Sort
Tempo para ordenar vetor DESordenado (10k): 2.530339002609253
Tempo para ordenar vetor DESordenado (25k): 17.27229928970337
Tempo para ordenar vetor DESordenado (50k): 74.87719321250916
Tempo para ordenar vetor ORdenado    (10k): 2.5474956035614014
Tempo para ordenar vetor ORdenado    (25k): 16.373104572296143
Tempo para ordenar vetor ORdenado    (50k): 78.17695569992065
###############################################
Bubble Sort
Tempo para ordenar vetor DESordenado (10k): 5.24332332611084
Tempo para ordenar vetor DESordenado (25k): 26.89321279525757
Tempo para ordenar vetor DESordenado (50k): 97.23622918128967
Tempo para ordenar vetor ORdenado    (10k): 3.7522547245025635
Tempo para ordenar vetor ORdenado    (25k): 23.37562394142151
Tempo para ordenar vetor ORdenado    (50k): 97.06031084060669
###############################################
```

```
Quick Sort
Tempo para ordenar vetor DESordenado (10k): 0.0140104293823242 19
Tempo para ordenar vetor DESordenado (25k): 0.0336556434631347 66
Tempo para ordenar vetor DESordenado (50k): 0.0664217472076416
Tempo para ordenar vetor ORdenado    (10k): 0.0100059509277343 75
Tempo para ordenar vetor ORdenado    (25k): 0.0287444591522216 8
Tempo para ordenar vetor ORdenado    (50k): 0.0484211444854736 3
###############################################
Merge Sort
Tempo para ordenar vetor DESordenado (10k): 0.0090036392211914 06
Tempo para ordenar vetor DESordenado (25k): 0.0263483524322509 77
Tempo para ordenar vetor DESordenado (50k): 0.0490047931671142 6
Tempo para ordenar vetor ORdenado    (10k): 0.0100047588348388 67
Tempo para ordenar vetor ORdenado    (25k): 0.0200042724609375
Tempo para ordenar vetor ORdenado    (50k): 0.0501739978790283 2
###############################################
Insert Sort
Tempo para ordenar vetor DESordenado (10k): 2.405769109725952
Tempo para ordenar vetor DESordenado (25k): 15.860849142074585
Tempo para ordenar vetor DESordenado (50k): 67.93823838233948
Tempo para ordenar vetor ORdenado    (10k): 0.00104522705078125
Tempo para ordenar vetor ORdenado    (25k): 0.0028822422027587 89
Tempo para ordenar vetor ORdenado    (50k): 0.0075635910034179 69
###############################################
```



Percebemos que sua Performance:
- No melhor caso (dados curtos): O(N log de N)
- No pior caso (dados longos): O(N log de N)

Qual a vantagem?
Um programa recursivo é mais elegante e menor que a sua versão iterativa e o mergesort é bem estável em diferentes situações.
Qual a desvantagem?
Ele é recursivo usa um vetor auxiliar durante a ordenação, ocasionando maior uso da memória.

Problema que isso pode gerar:
Se um algoritmo recursivo faz muitas chamadas, ele pode causar um estouro de pilha (stack overflow), ou seja, ficar sem memória suficiente para continuar a execução do programa.

Links utilizados:
https://dev.to/b0nbon1/understanding-big-o-notation-with-javascript-25mc
https://stackoverflow.com/questions/47973242/what-is-the-difference-between-pass-and-none-in-python
https://www.w3schools.com/python/ref_func_len.asp
https://realpython.com/len-python-function/#:~:text=The%20function%20len()%20is,of%20items%20in%20a%20list.
https://www.youtube.com/watch?v=5prE6Mz8Vh0&t=684s
https://docs.python.org/pt-br/3/tutorial/controlflow.html